An Improved Technique for Hierarchical Testing and Fault Simulation of Regular Circuits

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Dhruva Ranjan Chakrabarti

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

March 1996

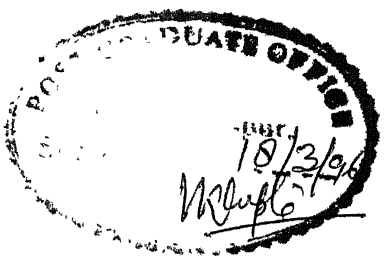
CSE-1996-M-CMA-IMP

CERTIFICATE

This is to certify that the work contained in the thesis entitled An Improved Technique for Hierarchical Testing and Fault Simulation of Regular Circuits by Dhruva Ranjan Chakrabarti has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. A. Jain,
Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur.

March 1996.



Acknowledgements

I am grateful to my thesis supervisor, Dr. A. Jain, for his guidance and encouragement throughout this work. He has always been eager to advise and help me in my need. I would like to thank Cadence Design Systems, India, for supporting this work in the form of providing me with a fellowship. I thank my friends, especially the M. Tech. '94 batch, for making my stay here a lively one. Finally I would like to acknowledge the support and encouragement of my parents, my brother and my sister throughout my stay here.

Abstract

An efficient scheme for developing test generators and fault simulators for *regular* circuits has been presented. An attempt has been made to show the usefulness of the property of *compatibility* between pseudo-sequential inputs and outputs of regular circuits. It has also been shown that test generators and fault simulators for regular circuits using the *bus fault model* and utilizing the concept of compatibility can be significantly faster than their gate-level counterparts. The approach is *hierarchical* since the algorithms start at a high level of abstraction and slowly unfold the modeled circuit while in pursuit of compatibility and consequent *parallelism*. The concept of *state transition graphs* has been used and it has been shown that attainment of compatibility is equivalent to obtaining a *loop* in the state transition graph.

The algorithm has been implemented to develop test generators and fault simulators for both combinational and sequential circuits. The results indicate a considerable speedup over conventional test generators for the regular circuits tested. A design technique for easily *testable and fault-tolerant* regular circuits has also been presented.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Rapid advances in circuit design and device fabrication processes in the last two decades have resulted in very large-scale integrated (VLSI) circuits containing millions of transistors. Such a large number of components in VLSI circuits has greatly increased the importance and difficulty of testing such circuits.

Designing of VLSI circuits has been simplified to a great extent by the development of systematic and computer-aided design (CAD) techniques that allow circuits to be designed and analyzed effectively at various levels of abstraction. However, the techniques for testing these circuits have not been developed at the same pace as others. While VLSI circuits are often designed and studied at a high level of abstraction, current test generation techniques usually require a low level model of the circuit. This leads to an enormous number of components in the circuit model and a consequent increase in the test generation time. Moreover, the possible advantages of the presence of repetitive sub-circuits and other important features remain unexplored during gate level test generation.

This problem was addressed by some researchers [BH90] by developing circuit and fault models at the high-level and formulating a hierarchical test generation procedure for combinational regular circuits. Though their test generator achieved a significant speedup over gate-level test generators, it did not have a complete fault coverage. In our work, we have tried to develop a methodology that uses the above model and achieves a better fault coverage for regular combinational circuits while

preserving the speedup over conventional gate-level test generators. The scheme has been extended to regular sequential circuits also. A hierarchical fault simulator which exploits the features of regular circuits and achieves considerable speedup over gate-level fault simulators has also been developed. Finally, a scheme for designing fault-tolerant regular circuits is presented.

In section 1.1 we define the basic terminology and the testing concepts used. Section 1.2 briefly surveys the relevant existing test techniques. Section 1.3 gives a brief outline of the presented technique while section 1.4 gives an outline of the rest of the chapters.

1.1 Background

Digital system testing deals with hardware faults, which are improper states of a digital system resulting from failure of components, physical interference from the environment, operator errors, or incorrect design [Avi75]. When there is a wrong output in response to a certain set of inputs due to a fault, the fault is said to produce an error. A set of input signals which causes the output of the circuit to be erroneous is said to detect a fault. The set of input signals is said to locate the fault if it is able to identify the faulty signal line.

The three basic steps in testing a digital circuit are test pattern generation, test pattern application and response verification. Input sets with ability to detect faults are generated by a test generator — these are applied to the circuit under test (CUT) and the responses of the CUT are verified by comparing them with known *good* responses. The CUT is faulty if its response does not agree with that of the correct value.

The fault model used usually at the gate-level is the single stuck-line model which assumes that any signal line in the CUT can be stuck at logical 1 or 0 while the circuit components (gates) are fault-free [Eld59]. The SSL model also assumes that at most one line in the CUT is stuck-at-0 or stuck-at-1 at a time.

1.2 Test Generation for Combinational Circuits

A majority of the test generation techniques for combinational circuits use a gate-level model of the circuit and the SSL fault model. Some of these are the D-algorithm, PODEM and FAN.

The D-algorithm was defined by Roth [Rot66]. The algorithm derives its name from the set of signal values S = $[0, 1, x, D, \overline{D}]$ used to describe the state of a line in the circuit during test generation. The assignment of value D to a line means that the line carries the signal 1 in the fault-free circuit and carries the signal 0 in the faulty circuit whereas the assignment of value $\overline{D}$ to a line means that the line carries the signal 0 in the fault-free circuit and the signal 1 in the faulty circuit. The value x denotes an uninitialized line. To generate a test pattern for a fault, the algorithm searches through the set of possible signal assignments to all the paths in the circuit such that the fault-effect can propagate through at least one of those paths to a primary output. This is called the sensitized path.

PODEM (Path oriented DEcision Making) [Goe81] directly searches the set of possible input patterns to obtain a test for a given fault. Assignments are hence made to primary inputs directly. It actually uses a branch and bound technique and manages to reduce the number of backtrackings compared to that of D-algorithm.

FAN [FS83], though based on the basic concepts of PODEM, uses various sophisticated techniques to reduce the number of backtracks and is significantly faster. SOCRATES [MMSS88, SA89, KP93] and Recursive Learning [KP94] are all based on the basic concept of FAN but use various learning techniques to speed up the test generation process.

Test vector generation for sequential circuits is considerably more complex than that for combinational circuits. Many of the conventional test generators for sequential circuits are based on methods available for combinational circuits and use a loop-free iterative array model of a sequential circuit. This is also called the time-frame expansion technique and seeks to generate a sequence of test vectors one time frame after another.

1.3 Outline of the Present Work

The present work deals with regular circuits, both combinational and sequential. The circuit and fault models (described in detail in Chapter 2) developed by Hayes and Bhattacharyya [BH90] have been used. The test generation method developed by them does not have complete fault coverage since the *incompatibilities* [BH90] between the interconnecting signal lines connecting various repetitive sub-circuits are not resolved. The approach developed by us tries to resolve incompatibility wherever possible and this leads to an enhanced parallelism in the test generation process and a consequent reduction of the test generation effort. The concept of the state transition graph has been used to solve the problem of incompatibility. This basic scheme has been applied to both combinational and sequential regular circuits with good results which show the feasibility of the approach. The same scheme has been used to develop techniques for efficient fault simulation and fault-tolerance.

1.4 Overview of the Other Chapters

Chapter 2 explains in detail the circuit and fault models and the basic test generation scheme used. Chapter 3 shows the application of the scheme to regular combinational circuits. Chapter 4 shows the application of the scheme to regular sequential circuits. Chapter 5 explains the fault simulation technique based on the same concept. Chapter 6 explains the design technique for regular circuits for easy fault-diagnosis and fault-tolerance. Chapter 7 concludes the thesis.

Chapter 2

The Test Generation Scheme

Structured design for testability techniques such as scan design [MAF90], along with powerful test generators such as those using Recursive Learning [KP94], have considerably eased the test generation problem for large circuits. However, these gate-level test generation algorithms [Goe81, FS83, MMSS88, CR94, KP94] employ test sets for logic gates considering them as basic building blocks of a circuit. But current design systems support hierarchical design consisting of conventional high-level modules such as adders, multiplexers, counters, decoders etc. along with repeated occurrences of identical sub-circuits which do not belong to any of the erstwhile mentioned conventional modules. In these hierarchically designed circuits, both conventional and unconventional high-level modules may be considered as basic building blocks and their features may be exploited to speed up the test generation process. While many existing high-level test generation algorithms [MH90, FSP85, SH81, TA80, BF80] make intelligent use of conventional modules by using precomputed test sets for them, they fail to exploit the repetition of unconventional modules. The motivation behind high-level testing is to avoid repetition of the test generation process for repetitive modules. However, many existing test generators are not able to accomplish this goal fully. The algorithm, presented in this thesis, takes into account the repetitive nature of unconventional sub-circuits, and attempts to reduce the overall test generation effort.

2.1 Circuit Model

The efficiency of test-generation for circuits with repeated sub-circuits can be improved, if the test vectors for the repeated sub-circuits can be generated in *parallel* by replacing them with a single module. The procedure adopted forms high-level sub-circuits by grouping together identical repetitive gate level sub-circuits [BH90]. *Buses* are formed from lines and the consequent circuit model consists of components such as word gates, multiplexers, adders, registers etc. connected with busses of appropriate sizes. The bus sizes are parameters reflecting the extent of *parallelism* exploited by the model, and may change depending on the level of abstraction. This generalization of a number of single-bit lines at the gate level to a single multi-bit bus at the high level gives rise to some unique modeling issues. Feedback loops are introduced at the high level though the circuit is combinational at the gate level. These loops are essential to represent the fact that identical modules spaced periodically in the gate level model are mapped onto the same high level model. In order to represent repeated sub-circuits as a single module, fanout (FO) and merge elements (ME) are introduced. Some of the primary inputs and outputs of the high-level model are called pseudo-sequential inputs (PSI) and pseudo-sequential outputs (PSO) respectively. The term psedudo-sequential arises from the sequential nature attained at an intermediate level of modeling and also from the fact that those inputs and outputs are not necessarily primary inputs and outputs at the gate-level. Some of the commonly used high-level components are shown in Figure 1.

2.2 Fault Model

The fault model, known as the bus fault model [BH90], is a generalization of the single stuck-at-line model [SSL] from single lines to busses and assumes that busses are stuck at some logic values. Since there is considerable experimental evidence indicating that a test set for all SSL faults provides very good coverage of the permanent faults encountered in practice, the SSL fault model has been used for estimation of fault coverage. This means that the fault coverage of our test generator is estimated in terms of the fraction of detectable SSL faults in the circuit.

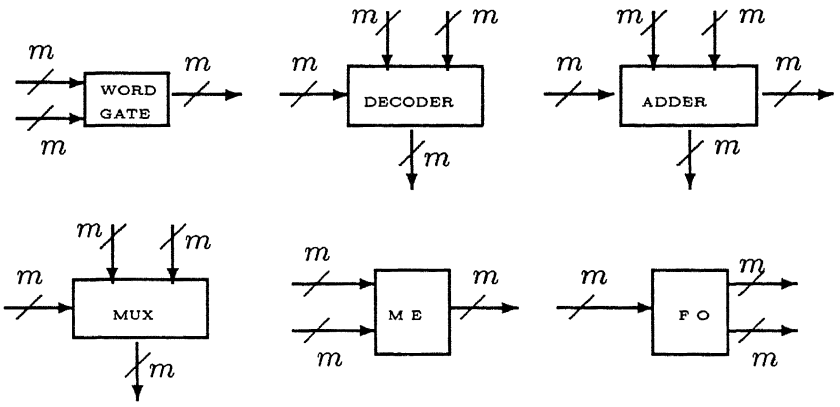
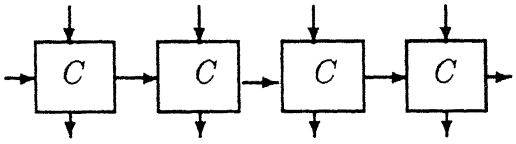


Figure 1: High Level Elements

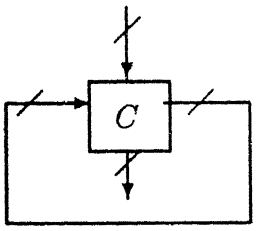
It has been shown in [BH90] that around 80% of the SSL faults in regular circuits in general can be detected by test generation using the bus fault model without resolution of incompatibility.

2.3 Test Generation Procedure

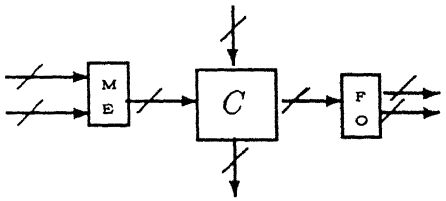
A simple class of **regular** circuits consists of *replicated modules* in a one-dimensional array with no intermodule connections. The low level model of such a circuit consists of m copies of an arbitrary gate-level model C. Our high level model for this circuit then consists of word gates and regular fanout elements only interconnected by busses of size m. This high level model of the circuit does not contain any pseudo-sequential inputs or outputs. This property makes test generation for the high level model extremely easy. Another class of **regular** circuits consists of replicated modules connected in a one-dimensional format. The high-level model in this case, however, possesses pseudo-sequential inputs and outputs which makes test generation more complicated than that in the previous case. Circuits which are not fully regular and consist of varying numbers of inputs in different modules can be made regular by adding additional signals. The test generator, however, has to take care of these additional signals by applying non-controlling values to them during



(a) Gate Level Circuit



(b) Modeled High-Level Circuit with feedback



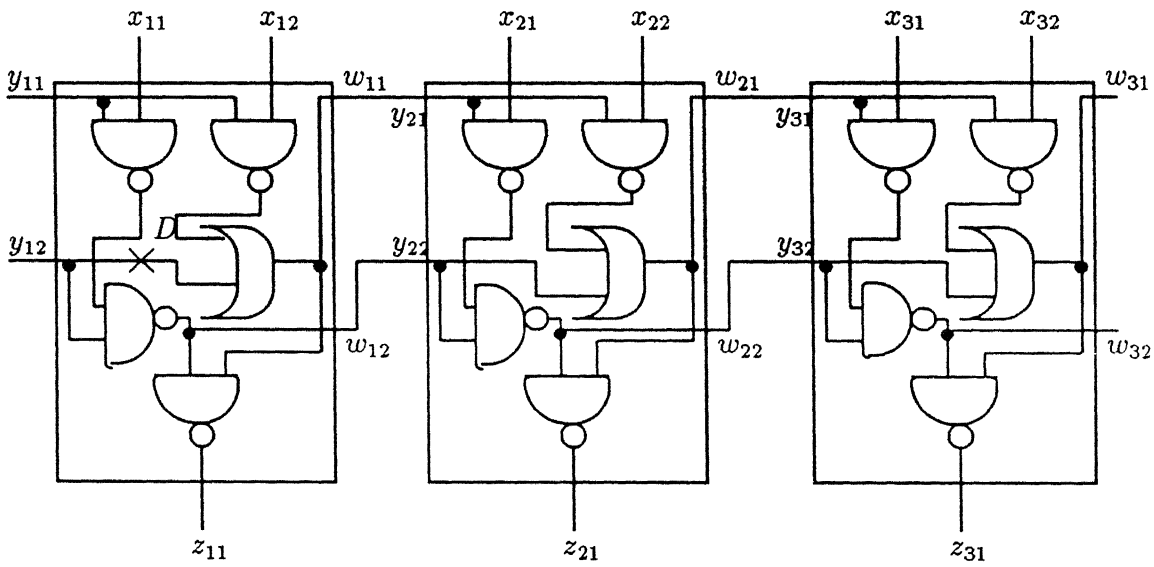
(c) Modeled High-Level Circuit without feedback

Figure 2: High Level Modeling

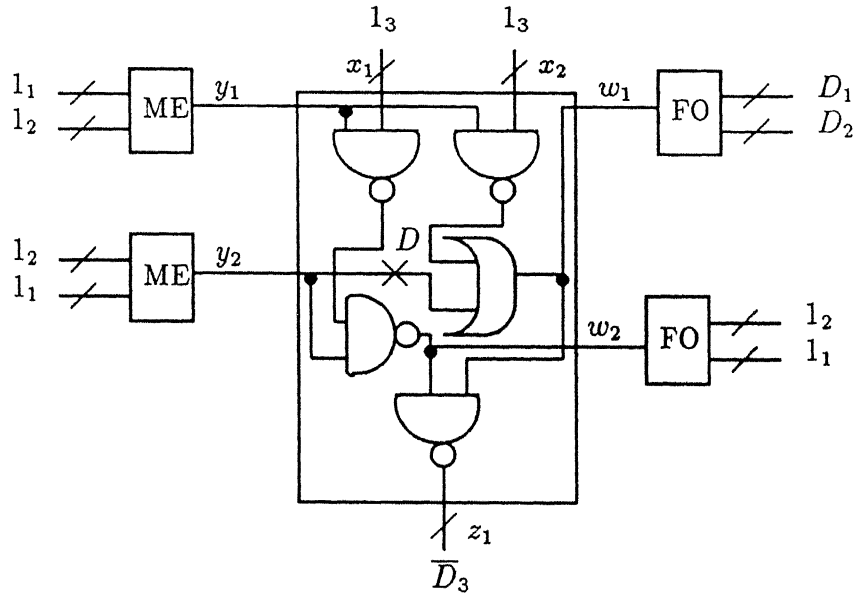
generation.

A pseudo-sequential contraction (PSC) procedure was developed in [BH90], which converts the gate level circuit to an intermediate high level pseudo-sequential (PS) circuit having a feedback path and ultimately to a purely combinational high level circuit with busses of appropriate sizes, fanout elements and merge elements. The final high level circuit has a reduced number of components which implies a reduced number of modeled faults. Moreover, the representation of the iterative sub-circuits is by a constant number of components with only the bus size varying. The number of components, however, changes with the level of modeling. A block diagram of the different stages of modeling is shown in Figure 2. While the number of components increases linearly in a gate level model with increase in circuit size, the bus fault model ensures a constant number of components with the bus size increasing linearly. The linear increase in the bus size does not pose any significant problem since the complexity of the algorithm is largely independent of the bus size. Figure 3 illustrates a gate-level circuit and the corresponding high-level circuit.

The test generation algorithm looks for test vectors for a bus fault in the high



(a)



(b)

Figure 3: An Example

level circuit such that the test vectors detect all SSL faults on that line in any one of the identical modules. These test vectors can be generated in parallel.

The algorithm has a number of advantages. First, the algorithm is independent of the abstraction and representation level of the used circuit and the fault models. Secondly, the grouping of identical modules into a single one and lines into a bus significantly reduces the number of target faults leading to a considerable reduction in the test generation effort.

One of the main issues to remember while generating test vectors using the discussed model is that the pseudo-sequential input busses are not directly controllable and the pseudo-sequential output busses are not directly observable. This means that simple propagation of the fault-effect to the PSO signal lines is not enough for generation of a valid test vector. It should be ensured that the fault-effect gets propagated to one of the primary outputs in the actual gate-level circuit.

Also, if we are to attain parallelism in test vector generation, the PSI and PSO signal lines should have compatible values. Two vectors V1 and V2, defined on the 5-valued set {0,1,D,$\overline{D}$,X} are said to be compatible [BH90] with each other, if replacing all occurrences of D and $\overline{D}$ in the two vectors by 1 and 0 respectively, results in vectors $V1'$ and $V2'$ such that $V1' \bigcap V2' \neq \phi$. In case of compatibility between PSI and PSO lines, the same input values can be assigned to the primary input of more than one sub-circuit. This leads to a significant reduction in the test generation effort.

But, in many circuits, during test generation for a fault, it is impossible to attain compatibility at the very top level. This actually means that the test vector is not of the form, $(x_1,....,x_n)_m$, where $(x_1,....,x_m)$ represents the values of the primary inputs in the faulty module. In such a situation, the circuit is said to be **pseudo-redundant** at the high-level. In this case, the algorithm should be able to look for compatibility after descending one level down the hierarchy. In the algorithm presented and implemented by us [CJ95b], we have tried to explore the possibility of using the bus fault model even if the circuit has been declared pseudo-redundant at the high-level. As an illustration, let us consider the circuit of Figure 4(a), and let us assume that the only test vector which can detect the fault in module 1 is 0111. The

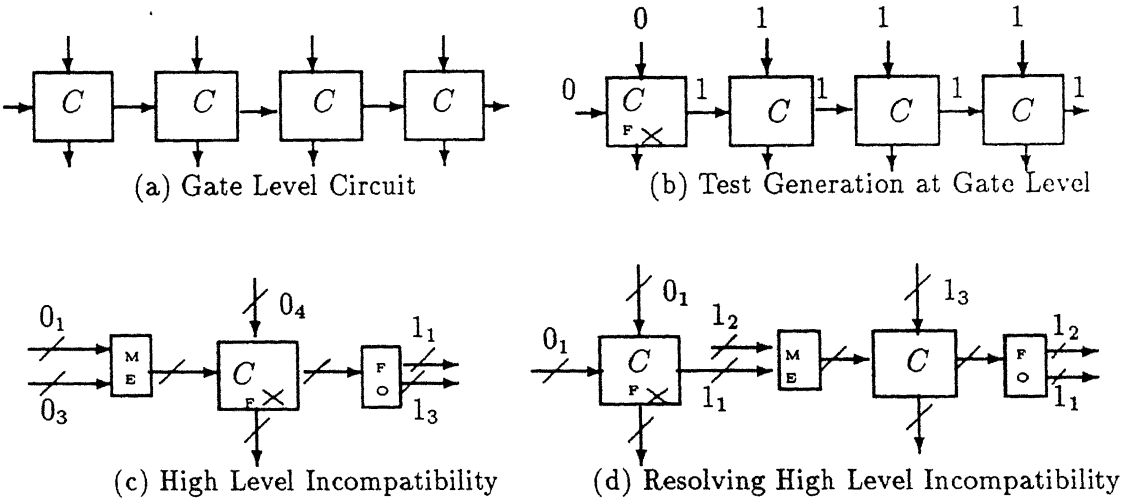


Figure 4: Solution of High Level Incompatibility

algorithm of [BH90] described above fails on encountering incompatibility, since the primary inputs required to be applied to all sub-circuits are not identical (the input for the first sub-circuit is 0, while the inputs to the other sub-circuits are 1). This has been illustrated in Figure 4(c). In this thesis, we have presented an algorithm to overcome the above problems in many cases where the above algorithm fails. The algorithm, initially, proceeds as described in [BH90]. However, on encountering an incompatibility, the high level circuit is unfolded one module at a time. While this searches the solution space implicitly and exhaustively, this also solves the incompatibility problem. Our algorithm succeeds in resolving incompatibility as has been shown in Figure 4(d).

2.4 Overview of the Test Generation Algorithm

The presented algorithm uses the high level circuit model and the bus fault model. In the most general case shown in Figure 2, each high level module has vector X = [x_1,....,x_n] as primary inputs and vector Z = [z_1,....,z_n] as primary outputs. The vectors Y = [y_1,....,y_p] (W = [w_1,....,w_r]) are the inputs (outputs) from (to) the other parts of the circuit. m denotes the bus size. A state transition graph (STG) can

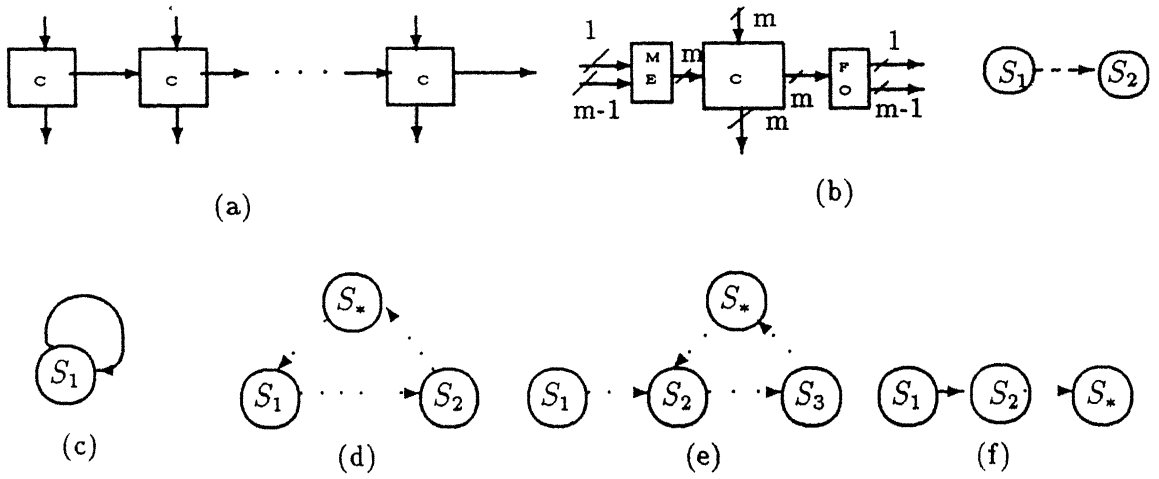


Figure 5: Possible State Transition Graphs

be drawn for this high level module, where the nodes represent the signal values Y or W. The arcs are labeled by signal values on X. As is shown in Figure 5, the transition from state S_1 to state S_2 has been effected by X= $[x_1,....,x_n]$. In the following discussion, $(x_1,....,x_n)_m$ denotes the input vector $(x_1,....x_n)$ repeated m times (one for each repeated sub-circuit).

Test for faults is generated using any existing automatic test pattern generation algorithm for combinational circuits like PODEM [Goe81], FAN [FS83] etc. If there is no incompatibility and the fault effect gets propagated to one or more of the Z-outputs, the test vector is automatically justified and the generated test vector is $(x_1,....,x_n)_m$. In this case, the state transition graph consists of a single node with an arc looping on it.

But if there is an incompatibility, it only means that the input vector and the output vector of the faulty module differ. This implies that justification of the assigned inputs in the faulty module is not yet complete and our next task is to justify the assigned vectors Y and W through fault-free modules. In this case, the state transition graph contains more than one node and the justification problem is equivalent to finding a loop in the graph starting from nodes representing vectors Y and W. If there is no loop present in the graph, the algorithm degrades to the

performance of a gate level algorithm. Heuristics are incorporated to find a loop in the state transition graph. In order to reduce both storage and computational complexities, only the necessary part of the state transition graph is scanned at any time.

The efficiency of the algorithm depends on the efficiency of the search conducted for a transition to one of the existing states of the state transition graph. This search should always start from the highest numbered state assuming that states are numbered (as they are obtained) in an ascending order. This is essential since a loop of smaller size is always better—this directly follows from the aim of preservation of the high level model and the consequent reduction of components and faults.

In the case of absence of a transition to one of the existing states, choice of the next state from many possible states is important and should be done in such a way so as to improve the chances of finding a loop in subsequent steps. Our approach tries to propagate the fault-effect to as many pseudo-sequential outputs as possible. This heuristic may not always be the optimal solution, but nevertheless, it is useful as has been shown by the results presented in the subsequent chapters. If the heuristic chooses the pseudo-sequential output values that do not lead to a solution, the algorithm ultimately backtracks and finds out the correct solution. Another possible heuristic may be choosing an *incompatible* vector for the pseudo-sequential outputs that has the *least distance* from the current highest numbered state. The success of this heuristic, however, shall depend on the components of the circuit. Algorithms can be developed for studying the parity of the various paths to a certain output of the PSI signal lines. This information may be successfully exploited to choose the bit to be made incompatible.

The approach is hierarchical in the sense that starting from a high level of abstraction, the algorithm slowly unfolds the circuit, i.e., it climbs down the hierarchy as incompatibility is reached. While coming down the hierarchy, the connectivity of the various modules should be preserved. This requires changing of the bus sizes, moving of fanout and merge elements and resetting the connections. The various steps involved in the algorithm are enumerated below.

2.4.1 Preprocessing

In the preprocessing phase, the functions implemented by the elements of the vector W are obtained in the following form.

w_i = f(y_1,....y_p,x_1,....,x_n); i=1,....,r.

These are later used to construct Boolean equations. Though the complexity of this problem is not polynomial, it is not likely to be prohibitive since the number of components and lines is greatly reduced for a single module. Also, this is a one-time job and the resulting Boolean equations may be stored for later use.

2.4.2 Handling Incompatibility

The test pattern generation process for the hierarchically modeled circuit proceeds in a way similar to the algorithm described before. After every change in assignment of signal values to a pseudo-sequential input [PSI] or a pseudo-sequential output [PSO], a check is carried out for compatibility between the PSI and the corresponding PSO. In case of incompatibility, normal execution of the test generator is suspended and it enters a different phase in order to resolve incompatibility. This phase acts only on the module having incompatibility. Thus, at present, the state transition graph of the incompatible sub-circuit has only two states, the initial state represented by [y_1,....,y_n] and the next state represented by [w_1,....,w_n].

Figure 6(a) shows the initial hierarchically modeled circuit. Since the obtained test vectors are required to cover all single stuck faults on that signal line in all identical modules, the final computation of the complete test vectors is done depending on the actual fault. So one of the following three cases can arise as shown in Figure 6.

(1) The fault is present in the first module (Figure 6(b)). In this case, the vector [y_1,....,y_n] is the output of some other sub-circuit and hence is not considered any further in this phase. The justification of this vector is done in the main routine. The level of the representation now changes—the high level module having signal values of bus size m is unfolded into two sub-circuits, one having signal values of bus size 1 and the other having signal values of bus size $(m-1)$. The signal values on the vector [w_1,....,w_n] of the sub-circuit having buses of size 1 is the same as the values

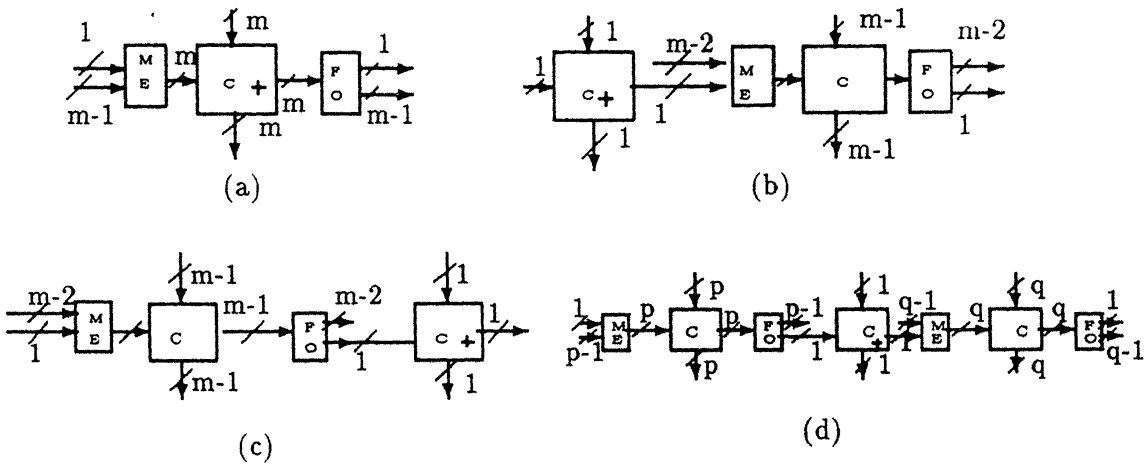


Figure 6: Possible Solutions of Incompatibility

on the vector $[y_1,....,y_n]$ of the sub-circuit having buses of size $(m-1)$. Starting with the vector Y of this second sub-circuit, a search is made for a transition to one of the two existing states. This is done by replacing the known values of $[y_1,....,y_n]$ and the target $[w_1,....,w_n]$ in the Boolean equations obtained during preprocessing and then solving them. In case of contradiction in all search attempts, a new state is obtained and the process is repeated.

(2) The fault is in the last module (Figure 6(c). This is similar to case 1, the only difference in the algorithm is the interchange of y's and w's.

(3) The faulty module lies in between the first and the last in the series (Figure 6(d)): This case is handled by separating the faulty module with buses of size 1 from two other sub-circuits, one having buses of size p and the other of size q, where $p+q=m-1$. The vectors $[y_1,....,y_n]$ and $[w_1,....,w_n]$ are then justified through fault-free modules.

Depending upon the relevant case, the main routine computes the test vector. In case the fault effect is not propagated to one of the primary outputs in the present module, it must be propagated to the next module in the series. So while finding a loop or a new state, the algorithm checks for this case. So while compatibility has to be achieved, it cannot be done at the expense of propagation of the fault-effect.

This is why the Boolean equation solver has to take care of 5-valued algebra. The Boolean equation solver uses a simple branch-and-bound technique. The results in the subsequent chapters indicate the feasibility of the approach.

This basic algorithm has been used to develop test generators for both combinational and sequential circuits with repetitive circuits. Chapter 3 describes this algorithm in the context of combinational regular circuits while chapter 4 explains the extension of this algorithm for sequential regular circuits. Chapter 5 describes a technique for fault simulation based on the same algorithm.

Chapter 3

Test Generation for Regular Combinational Circuits

The test generation algorithm for regular combinational circuits proceeds along the line of the algorithm described in the previous chapter [CJ95b]. The test generation algorithm has been developed over and above a conventional test generator. So apart from the usual modules, the other modules required are for handling incompatibility.

3.1 The Algorithm

The pseudo-code of the algorithm is given below.

```
preprocess()
begin
    for all outputs in vector W
    begin
        express output in terms of inputs
        Store this information
    end
end
```

```
/* end of preprocessing */

/* The following routine handles incompatibility. Since the gate-level circuit consists of m instances of the module, search for a loop is done at most (m - 1) times since one instance is already finished */
handle_incompatibility()
begin
    initialize the state transition graph
    for i = 1 to m - 1
    begin
        search_loop()
        if (search_loop() returns success)
        begin
            update the state transition graph
            return
        end
        find_new_state()
        if (find_new_state() returns failure)
            return(redundant fault)
        update the state transition graph
        update vectors Y and W
    end
end
/* end of handling incompatibility */

/* This routine searches for a loop to one of the existing states of the state transition graph */
search_loop()
begin
    set the search count
    for i = search count downto 1
```

```
    begin
        set the vectors Y and W
        solve_equation(Y,W)
        if (solve_equation() returns success)
            return the vector X with success
    end
    return failure
end
/* end of searching loop */


/* This routine finds the next state in the STG */
find_new_state()
begin
    propagate the fault from vector Y to vector W
    assign W to next state
end
/* End of finding new state */


/* This routine solves equations */
solve_equation()
begin
    solve Boolean equations using branch-bound technique
    if (no solution)
        return failure
    else
        return success
end
/* End of solving equations */
```

3.2 An Example

Figure 7 shows an example where an incompatibility is encountered and resolved. The actual gate-level circuit consists of m modules, identical to the one shown in Figure 7(a). We assume that the fault is present in one of the intermediate modules. It is clear from the position of the fault that the fault effect cannot be propagated to the primary output of the faulty module. So it has to be propagated to the signal lines W of the faulty module. Figure 7(a) shows the necessary assignments to vectors X and Y for the fault to be excited and propagated to the next module in the series. But in order to do so, an incompatibility has been encountered in Figure 7(a). Hence the level of abstraction is changed and the new high level model consists of three modules, the faulty module and the two fault-free modules on its either side.

Since the fault is in one of the intermediate modules, both the vectors W and Y will have to be justified for the fault to be propagated to one of the primary outputs. Figure 7(b) shows the justification of the vector W, obtained in Figure 7(a), through fault-free modules. Figure 7(c) shows the justification of the vector Y, obtained in Figure 7(a), through fault-free modules. Figure 7(d) shows the complete state transition graph. The final test vector obtained is $0(00)_p(11)(01)_q$, where $p+q = m-1$. It is clear that when m is large, this method gives a significant speed-up over gate level test generators.

3.3 Implementation and Experimental Results

The algorithm has been implemented in the C language on a DEC Alpha(DEC 2000). Since the algorithm requires both gate-level and high-level circuit descriptions to bring out its features, the conventional benchmarks which provide only a flattened netlist description have not been used to evaluate it. Either a translator should be provided or the benchmarks should be available in a hierarchical form in order to bridge the gap between circuit descriptions of conventional benchmarks and our present implementation. It has also been assumed that the locations of the repeated sub-circuits have been identified in the circuit description file.

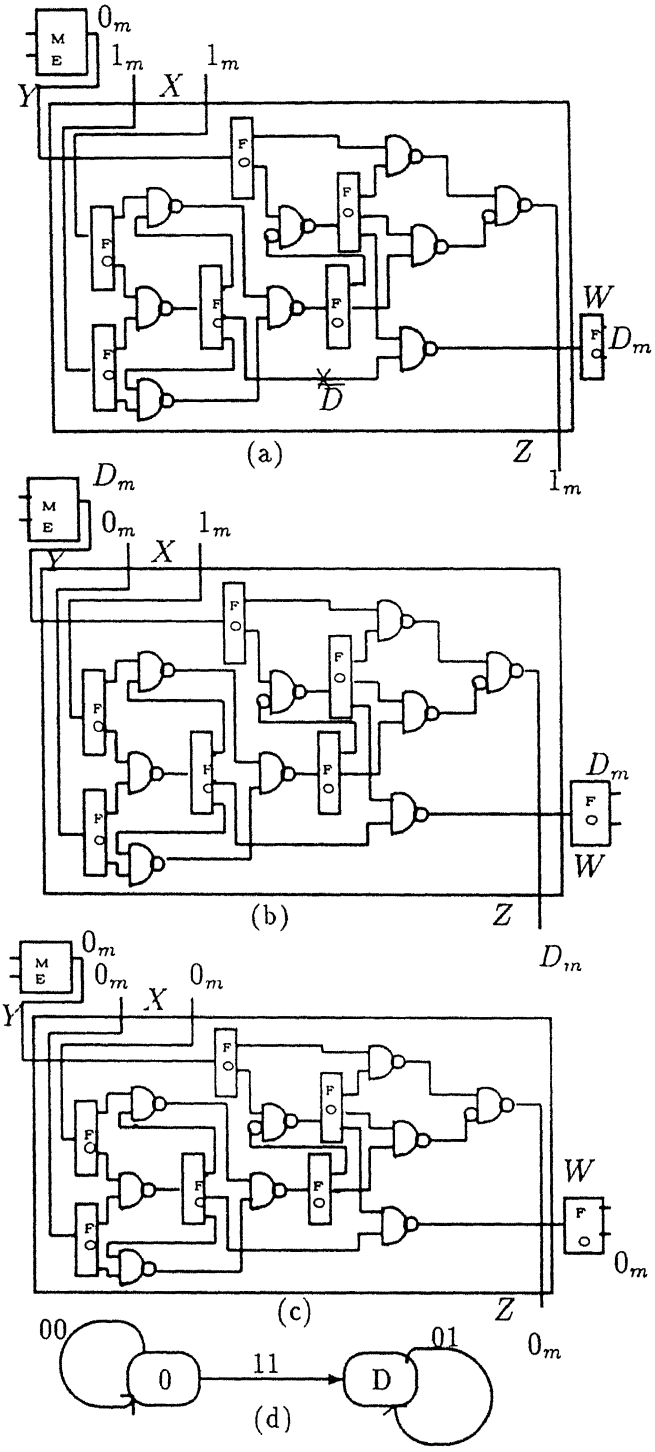


Figure 7: An Example

Ckt.	St	Ls	GL Algo	HB's Algo		Our Algo
			time	FC	time	time
cut1	1	20	1.6	100%	1.6	1.6
	4	71	28.4	45%	7.1	9.8
	8	139	417	45%	13.9	19.6
cut2	1	37	2.96	100%	2.96	2.96
	4	145	92.8	84%	17.4	23.2
	8	289	3468	84%	34.0	47.9
cut3	1	89	17.8	100%	17.8	17.8
	2	176	142.3	90%	70.4	105.6
	4	350	7013	90%	145	231.8
cut4	1	30	1.2	100%	1.2	1.2
	4	108	25.92	79%	6.48	7.56
	8	212	126.35	79%	13.72	18.9

Table 1: Combinational Test Generator : Comparative Results

PODEM is used as the basic algorithm for the three ATPGs, shown in the table and the graph. Search for compatibility is based on solving Boolean equations which have been modified to handle 5-valued algebra. The Boolean equation solver uses the concept of branch and bound technique; the resulting algorithm is simple and results indicate the feasibility of the approach.

Table 1 shows the performances of three algorithms, the conventional gate-level (GL), the technique mentioned in [BH90](denoted by HB), and our presented algorithm for four circuits. *St* indicates the number of stages, *Ls* indicates the number of lines. Results were obtained with different number of stages to observe its effect on the time required by the different techniques. The fault coverage (denoted by FC) of the gate-level test generator and our presented technique has been found to be 100% for irredundant faults and hence have not been listed in the table. The fault coverage of HB has been listed in the table. The time required by each technique in seconds is also shown in the table.

CUT 1 is a regular circuit. As Table 1 shows, the fault coverage without considering incompatibility is only around 45%. When incompatibility is resolved, fault

coverage reaches 100%. CUT 2 is a *modified* version of a ripple-carry adder shown in Figure 7. CUT 3 is a parity checker circuit. CUT 4 is based on the 74181 ALU. The gate-level and high-level descriptions of these circuits are available in [Dat85].

The time required to generate a test vector by our technique is almost *constant* for each fault irrespective of the number of stages. A substantial speedup over the gate-level test generator is obtained, especially, with increase in the number of stages. It has been seen (for the circuits listed in Table 1) that the loop size usually does not exceed 2 or 3. It was also observed that the number of times the modeled circuit has to be unfolded usually does not exceed 2.

Chapter 4

Test Generation for Regular Sequential Circuits

In this chapter, we present an extension of the test generation algorithm for regular sequential circuits [CJ96]. The chapter mainly deals with the modifications required for this class of circuits.

4.1 Extension of the Test Generation Algorithm to Sequential Circuits

The algorithm has been applied to sequential circuits with repetitive sub-circuits. The basic algorithm for finding test vectors for the modeled circuit remains the same as conventional test generators, the final sequence of test vectors being comprised of initialization, excitation and propagation vectors. Any of the existing sequential test generators such as [ea88] can be used for this purpose of finding the individual test vectors in the sequence for the high level model. But the problem of finding compatibility between PSI and PSO of a sub-circuit has to be solved in every time frame. The algorithm assumes that the flip-flops are *not* in a scan chain.

The salient features of the algorithm for sequential circuits are given below. A hardware reset is assumed to be available. However, the algorithm still works even if hardware reset is not available (in that case, number of time-frames in justification

phase may increase).

Preprocessing Phase :

Express each pseudo-sequential output as a boolean function of the pseudo-sequential inputs of the sub-circuit.

Excitation Phase :

Step 1 : Excite the fault using a single copy of the repeated sub-circuit.

Step 2 : If any of the flip-flops is not in the reset state, enter into justification phase, else go directly to propagation phase.

Justification Phase :

Step 1 : This phase, often referred to as the initialization phase, is intended to find the sequence of test vectors that would take the circuit from the initial state to the excited state. The algorithm tries to work its way from the excited state (where at least one flip-flop has its output unequal to the reset state) to the initial state (where all flip-flops have their outputs equal to the reset value). Initially, we start with a single copy of the repeated sub-circuit and steps (2) - (4) are carried out for each test vector in the justification sequence.

Step 2 : Check compatibility between PSI and PSO.

Step 3 : If incompatibility is encountered, the high level circuit is unfolded one at a time, and incompatibility is handled using STG as in case of combinational circuits [CJ95b].

Step 4 : Either compatibility is encountered (i.e., a loop is found in STG [CJ95b]) or the algorithm degenerates to the performance level of a conventional gate-level test generator.

Propagation Phase :

Step 1 : Propagate the fault-effect to as many PSOs as possible, considering a single copy of repeated sub-circuit.

Step 2 : If the fault-effect can not be propagated to any PSO, declare the fault to be redundant.

Step 3 : If the fault-effect is propagated to only the feedback paths, check if any of

the non-feedback PSO depends on the feedback paths to which the fault effect has been propagated. If there is no such dependence, declare the fault as redundant.

Step 4 : Check for compatibility in non-feedback lines. If incompatible, call routine for handling incompatibility which unfolds the circuit one sub-circuit at a time to find a loop in STG [CJ95b].

Step 5 : Either compatibility is encountered, i.e., a loop is found in STG [CJ95b], or the algorithm degenerates to the performance level of a conventional gate-level tnest generator.

The ultimate loop size will be the maximum of loop sizes found in justification and propagation phases. Figure 8 illustrates the algorithm for test vector generation for sequential circuits. The original circuit is first converted into the model appropriate for application of the proposed testing algorithm. For simplicity, the bus sizes, FO elements and ME elements are not shown in the figure. The algorithm starts with a single copy of the sub-circuit. Excitation of the fault, shown in Figure 8(a), and assignment of values to inputs so as to propagate the fault effect to the outputs results in X_1 = 'x', X_2 = '0', Y_1 = 'x', W_1 = '1' and Z_1 = '1'. Thus compatibility is attained in this time frame and the STG consists of a single state with a self-loop. Since the fault effect is not propagated to a primary output but only to the feedback line, the next time frame has to be considered. The second time frame also starts with a single copy of the sub-circuit and attempts to propagate the fault effect to a primary output. This results in X_1 = 'x', X_2 = '0', Y_1 = '1', W_1 = '$\overline{D}$' and Z_1 = '$\overline{D}$' in the second time frame. Thus the fault effect gets propagated to a primary output in the second time frame but incompatibility is encountered. To solve this incompatibility, the hierarchy of the model is changed and the original modeled circuit is unfolded to form two sub-circuits as shown in Figure 8(b). A search for a loop is conducted and a loop is obtained in the second time frame of size 2. This is because with X_1 = 'x', X_2 = '0', X_3 = '0' and X_4 = 'x', we get Y_1 = '1' and W_2 = 'D'. Therefore Y_1 and W_2 are compatible. Thus the final size of the loop is MAX(1,2) = 2. The test sequence obtained is $x(x0)_m$, $1(x00x)_{m/2}$, assuming a series of m identical sub-circuits.

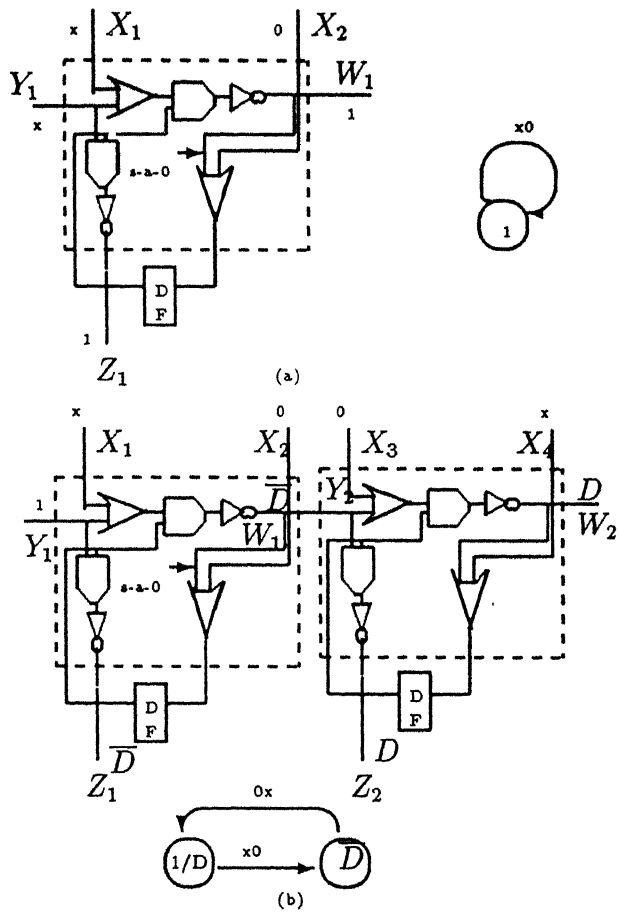


Figure 8: An Example

The example illustrates the method used in obtaining the test vector for a sequential circuit. There is no limit on the number of lines connecting two identical sub-circuits. Every sub-circuit need not have a primary output. This will mean that the fault effect must be propagated through the series of modules The present implementation is able to handle this problem of propagation of the fault effect through intermediate fault-free modules.

Ckt.	# Lines	# F/F	GL Algo Time(s)	HL Algo Time(s)
SN74S163	98	4	8.95	3.19
SN54LS160A	163	4	19.56	7.22
SN54LS190	115	4	4.79	1.65
SN54LS299	250	8	40.91	6.14

Table 2: Sequential Test Generator : Comparative Results

4.2 Implementation and Experimental Results

The proposed algorithm has been implemented in the C language on a DEC Alpha (DEC 2000). Since the algorithm requires both gate-level and high-level circuit descriptions to bring out its features, the conventional benchmarks which provide only a flattened description, have not been used to evaluate it. Either a translator should be provided or the benchmarks should be available in a hierarchical form in order to bridge the gap between circuit descriptions of conventional benchmarks and our present implementation. While our proposed algorithm is for circuits with repetitive sub-circuits, the conventional benchmark circuits usually do not possess this feature and this is another reason why they have not been used. It has been assumed that the locations of the repeated sub-circuits have been identified in the circuit description file.

The circuits listed in Table 2 are taken from [Dat85]. The first three circuits are various types of counters while the fourth one is a shift register. The number of lines and flip-flops in each of the circuits have also been tabulated. The number of flip-flops indicates the number of repetitive sub-circuits present in a circuit. As is shown in the table, a significant speedup is obtained for each of the circuits. However, in each of the cases, the speedup is less than the number of repetitions owing to the overheads involved in handling incompatibility, whenever self-loop is not obtained. The time required to generate a test vector using our technique is almost constant irrespective of the number of repetitive sub-circuits present in the actual circuit. This is why the speedup is more significant usually when the number of repetitive

sub-circuits increases. For example, test generation of a modified SN54LS299 after extending it to 16 repetitive sub-circuits has been seen to yield a speedup of more than 25. The fault coverage of the gate-level test generator (GL) and our proposed technique (HL) have been found to be 100% for irredundant faults for the circuits tested and hence have not been listed in the table.

The basic algorithm used for the two ATPGs, shown in the table, is a modified version of the iterative array or time-frame expansion algorithm [MAF90]. Search for compatibility has been based on solving Boolean equations which have been modified to handle 5-valued Boolean algebra [CJ95b]. The Boolean equation solver uses the simple branch and bound technique. Though the solution of Boolean equations has an overhead associated with it, the time required by the technique never increases exponentially in terms of primary inputs since the loop-size has usually been found to reach a maximum of 3 for the repetitive circuits tested.

In the present implementation, the preprocessing phase and the construction of the high-level model have not been incorporated, i.e., they have not been automated. The construction of the high-level circuit should take into account slight differences in repetitiveness of the sub-circuits and modify the original circuit accordingly. The time shown in Table 2 *does not* include that required for preprocessing which is a one-time job for any circuit.

Chapter 5

A Fast Fault Simulation Technique for Regular Circuits

Fault simulation is usually an important part of the test generation process for logic circuits. It may be used for various purposes, such as determining the fault-coverage achieved by a set of tests, analyzing behaviour in the presence of faults etc. It is often used along with the test generation process. As a test vector is generated, the faults are simulated using the test vector, and the detected faults are dropped from the fault set. This avoids the generation of too many tests for any one fault in the circuit.

The typical input to a fault simulator thus consists of a circuit model, a list of input patterns and a list of faults applicable to the circuit. Fault simulation consists of simulating a circuit in the presence of faults. Comparing the fault simulation results with those of the fault-free simulation of the same circuit simulated with the same applied test vector T, we can determine the faults detected by T.

A lot of research has gone into the development of efficient fault simulators. However, the speed of these simulators can be improved if these take the regularity of the circuit into consideration. The algorithm proposed by us tries to speed up the fault simulation technique by exploiting the features discussed in the previous chapters.

5.1 The Fault Simulation Algorithm

The fault model employed is the single-stuck-at fault which means that the algorithm handles one fault at a time. The algorithm works best if the test vector applied for simulation has *loops*. The test generation method described previously always tries to choose such vectors, if possible. In absence of loops in the test vector applied to the regular circuit, the performance of our fault simulator, however, degenerates to the performance of a conventional fault simulator.

The main steps of the algorithm are as follows :

Step 1 : Check for presence of loops in the test vector. If there is no loop, perform an ordinary fault simulation. Otherwise, proceed to step 2.

Step 2 : If the loop in the test vector does not comprise of the sub-circuit having the faulty signal, proceed to step 4 else proceed to step 3.

Step 3 : This step will be executed if the loops in the test vector (applied to the circuit) envelope the sub-circuit having the faulty signal line. First, an ordinary simulation of the sub-circuits towards the left of the sub-circuit corresponding to the start of the loop is carried out. Then simulation of the sub-circuit having the faulty signal line is carried out. If the fault-effect gets propagated to the pseudo-sequential output of the sub-circuit, then compatibility of the PSO lines is checked with that of PSI lines. In case of compatibility, no simulation is required for the sub-circuits towards the primary output of the faulty circuit. The simulation process can stop at this point, signaling detection of the fault. Thus this method achieves speedup over conventional methods since it does not need to simulate all portions of the circuit. If the fault-effect does not get propagated to the faulty sub-circuit, the simulation process terminates signalling non-detection of the fault since clearly that fault will never be detected by the test vector under consideration. Our algorithm can take this decision since it utilizes the structural information of the circuit, namely the repetitiveness of the circuit and the interconnections among various sub-circuits.

Ckt.	Av. % of faults not Requiring Complete Simulation
Cut 1	78
Cut 2	61
Cut 3	64
Cut 4	59

Table 3: Simulation Results

Step 4 : If the loops in the test vector envelope sub-circuits towards the primary input of the faulty sub-circuit, then simulation of those sub-circuits can be speeded up by checking for compatibility. In case of compatibility, all the sub-circuits need not be simulated since the PSO signals of the various sub-circuits will be identical. If the loops in the test vector envelope sub-circuits towards the primary output of the faulty sub-circuit, the propagation of the fault-effect (towards the primary output), if any, can be speeded up in case of compatibility between PSI and PSO signal values.

5.2 Implementation and Results

A simplistic implementation of the algorithm has been carried out. Any conventional algorithm may be used to simulate the individual high level modules. The results for some of the regular circuits (taken from [Dat85]) show that for a large percentage of faults, simulation of the complete circuit was not necessary. The circuits are the same as those described in the earlier chapters. Table 3 shows the average percentage of faults for which complete simulation of the circuit was not necessary. This number has been seen to range from 50% to so high as 90% in some individual cases. This indicates significant speedup in the process of simulation.

Chapter 6

Design of Fault-tolerant Regular Circuits

Many VLSI circuits like adders, multipliers, decoders, parity checkers etc. consist of identical modules connected in a regular repetitive fashion. In presence of faults, such circuits are usually rendered unusable. However, a design consisting of a number of standby spare sub-circuits (identical to the repeated sub-circuit) and augmented with some redundancy capable of diagnosing and bypassing the faults will contribute towards the longevity of the usefulness of the chip. This chapter focuses mainly on the diagnosis and reconfiguration of the repeated sub-circuits so that the faulty chip continues to function properly [CJ95a]. The chapter also discusses the overheads both in terms of extra hardware and additional time needed for fault-diagnosis and reconfiguration.

6.1 The Scheme for Fault-Diagnosis and Tolerance

As discussed in previous chapters, the loop-sizes of the STG during test generation are usually small and the level of unfolding not very deep, the signal values at PSI or PSO repeat themselves at regular intervals in a fault-free circuit. Thus instead of computing the signal values at each and every output of the circuit under test

and then comparing them with fault-free values, we can simply check for repetition of PSI or PSO if the repeatability is known beforehand. It may be appreciated that this method is effective only if the test vectors are of a repetitive nature. Thus if a fault is detectable by more than one input vector, the one with more repetitiveness would be the *right* vector to choose for the diagnosis phase. The test generation procedure described above always tries to choose the *right* test vector from a set of valid test vectors. This aspect has provided the motivation to look for a new scheme for fault-tolerance using the above concept.

The fault model employed here is the single stuck-at-fault model. We look for test vectors capable of exciting and propagating a fault to the output of a repeated sub-circuit and also having the capability to form a loop.

The possible state transition graphs described before have been reproduced in Figure 9 for convenience. Figure 9(c) shows a vector of loop-size 1, i.e., a self-loop. Figure 9(d) shows a vector of loop-size n (n > 1). Figure 9(e) shows a vector which does not have a loop to the initial state but has a loop to an intermediate state, whereas Figure 9(f) shows a case where there is no loop. When test vectors with ability to form a loop are applied to the inputs of a circuit, some lines are found to be compatible in a fault-free circuit. If a fault detectable by the applied test vector is present, incompatibility in those lines is detected by the comparison circuitry and reconfiguration is performed. Figure 9(g) shows the modified circuit in the form of a block diagram. In Figure 9(g), the repeated sub-circuit is denoted by C.

In order to obtain easy testability and a fault-tolerant design, some additional circuitry is needed in the form of multiplexers and logic gates. The 2-to-1 multiplexers M_* determine the source of inputs to a sub-circuit and are used for bypassing sub-circuits. The 2-to-1 multiplexers N_* determine the signals to be compared for compatibility. These signal lines define the boundary of the *test-window* at any point in time. The control signals of the multiplexers have not been shown in the diagram for simplicity. When the control signal of a multiplexer is '0', the input coming from the corresponding sub-circuit is passed to the output, whereas the output from the previous multiplexer is selected if the control signal is '1'. X represents an array of EX-OR gates followed by an OR gate. Two states of an STG are compatible

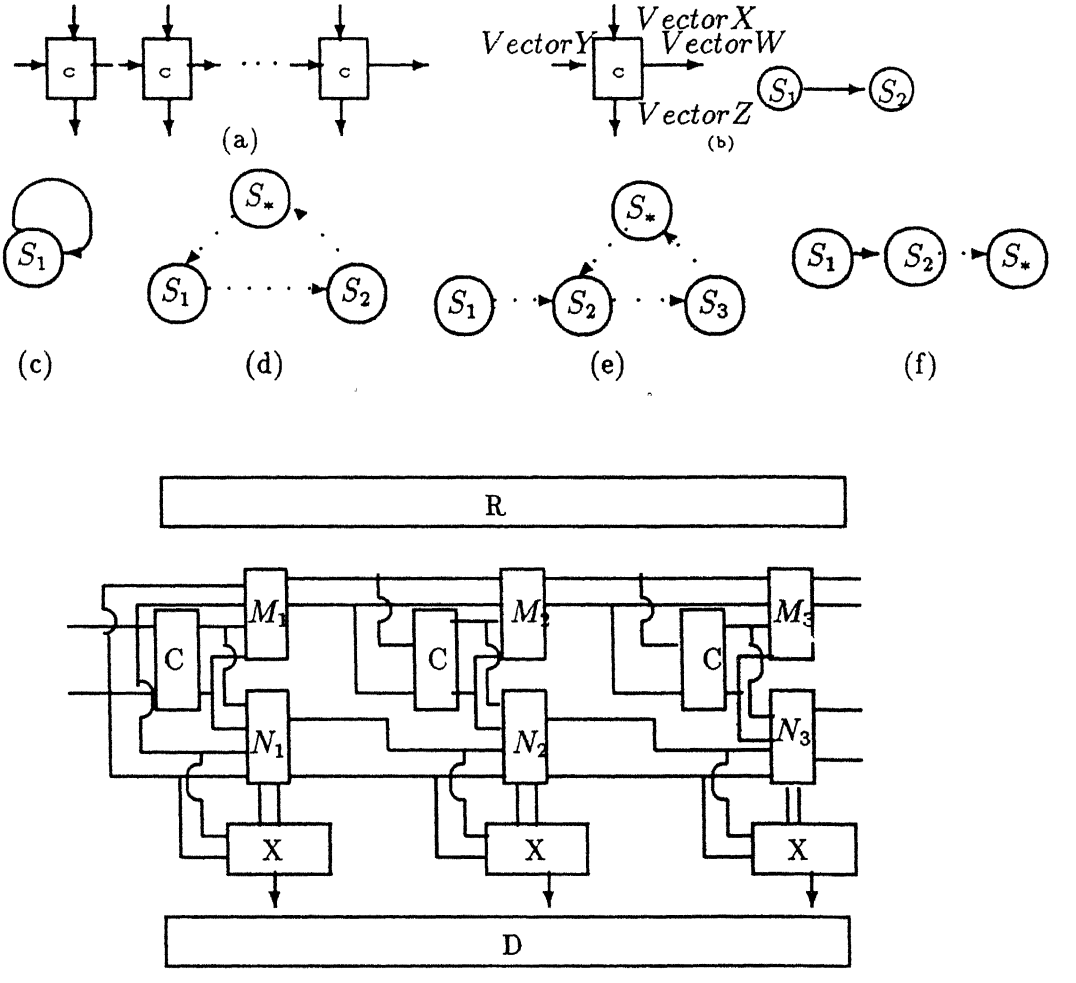


Figure 9: The Modified Circuit

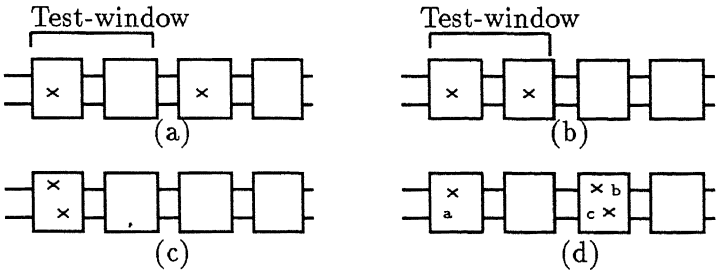


Figure 10: Multiple Faults

only if all the lines are individually compatible. The outputs of all EX-OR gates corresponding to a sub-circuit are fed to an OR gate. The output of the OR gate indicates the result of comparison of the states — '0' denotes compatibility, whereas '1' denotes incompatibility. D and R are the diagnosis register and the reconfiguration register respectively. The length of these registers equals the number of repetitive sub-circuits. For each signal line interconnecting two successive sub-circuits, two multiplexers are required. An array of EX-OR gates and an OR gate are required for each sub-circuit. The number of EX-OR gates for each sub-circuit is equal to the number of lines interconnecting two successive sub-circuits.

Though the basic fault model employed is the single stuck-at-fault, the same algorithm is expected to give error-free operation in presence of multiple faults, as can be seen by examining some possible cases as follows.

Case 1 : Faults in the same module

This situation has been shown in Figure 10(c). If the faults mask each other, the circuit is expected to operate without any error. If the faults don't mask each other, any non-masked fault will be detected and the module bypassed.

Case 2 : Faults in different modules

Let us consider two faults at a time. If the faults are not encompassed by the *test-window* (Figure 10(a)), then the scheme is effectively similar to diagnosis of single faults since each of the multiple faults are considered one at a time. If the faults are encompassed by the *test-window* (Figure 10(b)), then either of the following two cases may arise. If the faults mask each other, the circuit is expected to coninue operating in an error-free manner. If the faults do not mask each other, then each

of the faults will be located as the window is shifted one module at a time.

Case 3 : Other Faults

This situation is the simultaneous occurrence of the previous two situations and the same scheme is expected to succeed here also. However, a particular situation deserves discussion. Let us assume that a, b and c are three faults occurring simultaneously as shown in Figure 10(d). Let a and b mask each other while c be independent of the other two. So faults a and b will not be detected while fault c will be detected. But subsequently, when module number 3 is bypassed owing to detection of fault c, fault a becomes manifest and hampers the normal operation of the circuit. Our scheme takes care of this situation by repeating a pass if the previous pass led to bypass of some module. It may be inferred that in the worst case, the number of passes is equal to the number of faults present which is finite owing to the usual limit on the number of spare sub-circuits.

6.2 The Reconfiguration Algorithm

6.2.1 Test Generation Phase

(1) The sub-circuit being repeated is preprocessed to find the test vectors for every fault.

(2) For each of these test vectors, the existence of a loop is checked. Details regarding search for a loop are given in [CJ95b].

(3) The test vectors are then ordered on the basis of smallest loop-size and maximum fault coverage with the former as the primary key. During the diagnosis phase, the test vectors are applied in this order till maximum fault coverage is obtained.

6.2.2 Fault Diagnosis Phase

Case 1: Test vectors having loop-size 1

(1) The control signals of all multiplexers M_* are set to 0, thus forcing inputs of a module to come from the outputs of the previous module.

(2) The control signals of all multiplexers N_* are set to 0, thus forcing comparison between input and output signals of every module.

(3) The outputs of the OR-gates are stored in a shift register.

(4) If all bits in the shift register are '0', no fault has been diagnosed and no reconfiguration is required; otherwise the reconfiguration phase is entered, which consists of the following steps.

(5) The reconfiguration register is loaded from the diagnosis register.

(6) The i^{th} bit of the reconfiguration register is the control input to the multiplexer M_i.

Case 2: Test vectors having loop-size n (n > 1)

(1) The control inputs of all multiplexers M_* are set to 0. The control inputs of all multiplexers N_* are initially set as follows :
N_1,...,N_{n-1} = '1', N_n = '0', N_{n+1},...,N_{2n-1} = '1', N_{2n} = '0'.....

(2) The outputs of the OR-gates are stored in the diagnosis register.

(3) If the diagnosis register contains all zeros, no reconfiguration is required since no fault has been diagnosed.

(4) If there is a fault, which can be detected by the test vector, in any one module in a group of n modules, the first (n-1) bits of that group will be 0 and the n^{th} bit will be 1. Let us consider this group as the faulty window in the diagnosis register. The actual faulty module is located by replacing one module at a time from the faulty window with one from a fault-free group. This uses several passes as described below. The n-bits of the reconfiguration register corresponding to the faulty window is loaded with S, where S= (100....00) initially. This means that the first module from the left of the faulty window is being bypassed in this first pass.

(5) The control signals given to multiplexers N_* are shifted 1 bit to the right with data input 1 at the left. This shifting is done once, only in the first pass. The faulty window is shifted only one place to the right. Since we are considering only single faults, the fault-free sub-circuit to the right of the faulty window is used to replace one sub-circuit from the faulty window at a certain pass.

(6) The outputs of the OR-gates are stored in the diagnosis register.

(7) If the diagnosis register contains all zeros, the rightmost sub-circuit just left of the current window is the faulty module. The fault gets located and the reconfiguration register should be loaded appropriately as in the previous case of self-loop.

(8) If all bits in the diagnosis register are not zero, a right shift is applied to S with data input '0' at the left and the process is repeated by going to step (5).

Case 3: Test vectors of the type shown in Figure 9(e)

The number of transitions to the state from which the loop starts is counted. Also the number of transitions starting from the first state to the end of the loop is counted. Let these two counts be denoted by n and m. The control signals of the multiplexers N_* are set as shown below.

N_1,...,N_n = '0', N_{n+1},...,N_{m-1} = '1', N_m = '0'. The faulty window is from bit (n+1)

to bit m. So while shifting, only the bits in the faulty window and the next successive bit should be manipulated. The other steps are similar to that for test vectors of loop-size n, where (n>1).

Case 4: Test vectors of the type shown in Figure 9(f)

In this case, the test generation phase does not achieve any speedup as compared to other methods since the special properties of a repetitive circuit cannot be used. However, the diagnosis phase locates the fault by bypassing sub-circuits one at a time and then appropriate reconfiguration as in the previous cases is performed. Figure 11 shows a circuit consisting of a number of repetitive sub-circuits C. The sub-circuit C, which constitutes one block of the 74181 ALU [Dat85], has been shown in Figure 11. Let us assume that all the four sub-circuits together form the original circuit. The spares are not shown for simplicity. The faults have been shown in Figure 11. The test vector for faulty line # 2 is $V_2 = 00(111100)(111000)_3$ and that for faulty line # 1 is $V_1 = 10(111000)_4$. When V_2 is applied , according to the algorithm described above, the control signals of all multiplexers are set to zero. Since the test-window is comprised of sub-circuit C_2, the contents of the diagnosis register will be $(d1dd)$ where d denotes don't care binary value. In the next phase of the algorithm, a shift is applied and the diagnosis register will contain $(d0dd)$. The difference in the value in the second bit position of the diagnosis register indicates that the second sub-circuit is faulty. It is assumed that a fault-free spare replaces the faulty sub-circuit with appropriate reconfiguration as described in the algorithm before. When V_1 is applied, the control signals of the multiplexers are set to zero and the diagnosis register contains $(ddd1)$. Since this is the same as case 1 of the algorithm above, we conclude that the fourth sub-circuit is faulty. The faulty module is subsequently bypassed.

6.3 Results

The algorithm was run on a number of circuits, each with a few spares, after injection of faults. The results are shown in Table 4. Cut 1 is a circuit which has

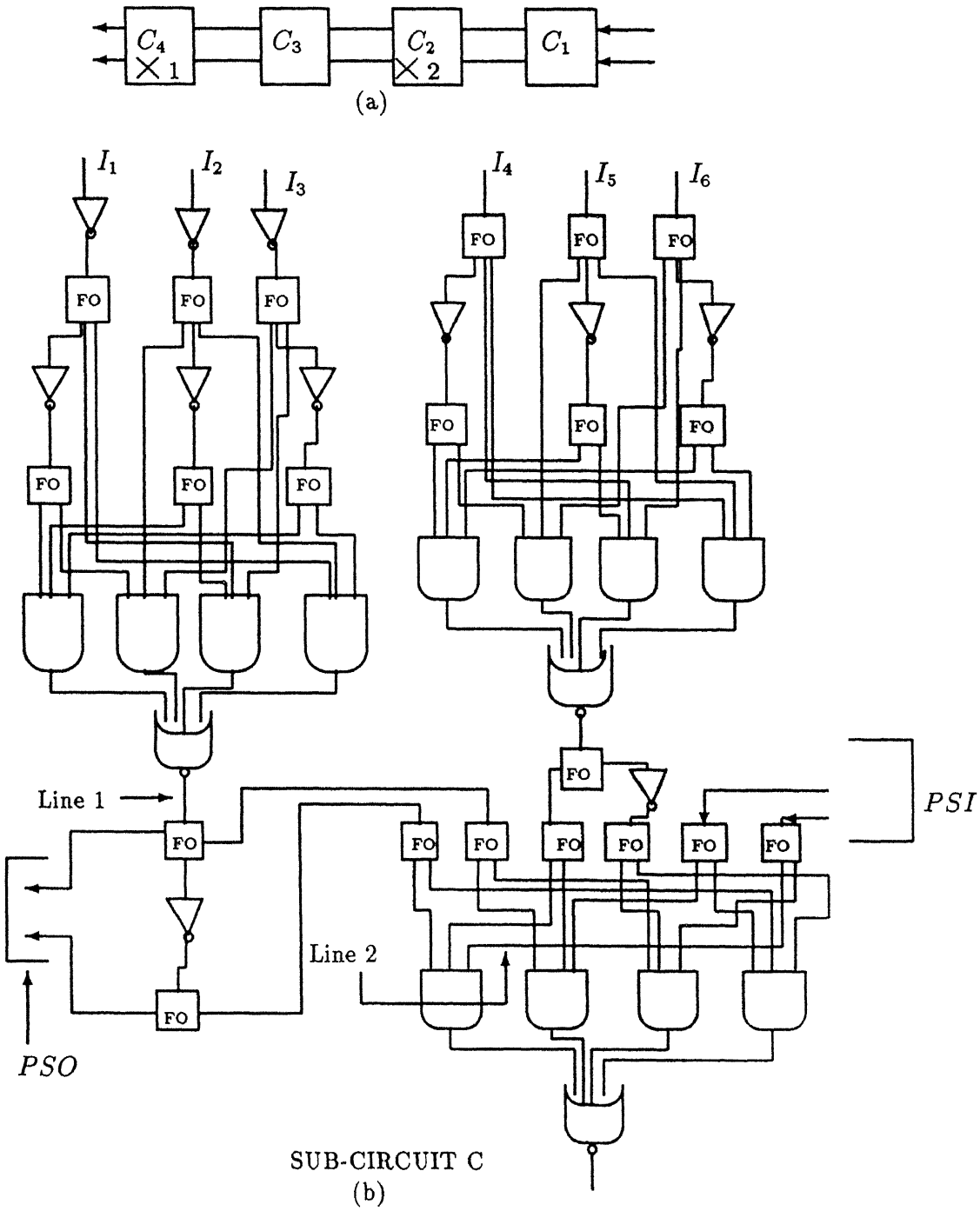


Figure 11: An Example

Ckt.	Number of Stages	Number of Lines	Number of Faults	Number of Spares	Average Number of Passes
Cut 1	16	289	8	3	2.2
Cut 2	32	1155	12	6	4.8
Cut 3	16	1520	14	3	2.7
Cut 4	32	968	10	6	3.9

Table 4: Reconfiguration Results

been specially designed to bring out the salient features of our algorithm. Cut 2 is a modified version of a ripple-carry adder. Cut 3 is a parity checker circuit while Cut 4 is based on the 74181 ALU. The faults have been injected randomly with the only restriction that the number of faulty modules does not exceed the number of spares. The average number of passes is based on around 100 simulations and exceeds one indicating that multiple faults are diagnosed.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we have presented the test generation algorithm for combinational and sequential regular circuits. A fault-simulator based on this algorithm has also been discussed. An attempt has been made to show the usefulness of the property of compatibility between pseudo-sequential inputs and outputs of regular circuits. It has also been shown that test generators and fault simulators of regular circuits using the bus fault model and utilizing the concept of compatibility can be significantly faster than their gate-level counterparts. The approach is hierarchical since the algorithms start at a high level of abstraction and slowly unfold the modeled circuit while in pursuit of compatibility and consequent parallelism. The concept of state transition graphs has been used and it has been shown that attainment of compatibility is equivalent to obtaining a loop in the state transition graph. It has been found for the circuits tested that the loop sizes are usually small and the level of unfolding of the initial modeled circuit not very deep. Various heuristics have been suggested to solve the various problems involved and keep the complexity in check.

The algorithm has been implemented to develop test generators for both combinational and sequential regular circuits. The results indicate a considerable speedup

over gate level test generators for the circuits tested. A fast fault simulator, utilizing the features of repetitive circuits has been implemented. The results show the feasibility of the approach. Finally, a design technique for easily testable and diagnosable regular circuits has been presented.

7.2 Future Directions

The current work seeks to propose new techniques for testing and simulating regular circuits (both combinational and sequential) and show the feasibility of the techniques. However, further work can be done to increase the efficiency and scope of the techniques used. Some possible directions for this are now discussed.

A basic assumption of our implementation is that both the high and the low-level models of circuits under test, and the locations of the repeated sub-circuits, are known. However, present benchmarks provide only a gate-level model of the circuit. This requires automatic identification of the regularity in the gate-level circuit and generation of the high-level circuit replete with high-level modules like fanout elements, merge elements, buses of appropriate sizes and the like. Our implementation takes a simplistic view of this situation and requires the user to provide all this information. Though the general problem seems to be intractable, work may be carried out to improve upon the user-interface and move part of the onus of development of the high-level model from the user to the program. A solution for a particular set of circuits may also be explored.

The Boolean equations may be represented in some alternative form like binary decision diagrams (BDDs) which may facilitate faster processing. Also, the presented concepts can be utilized in designing a scheme for synthesis for testability.

The fault simulator implemented by us does not use any sophisticated simulation algorithm. Work may be carried out to develop a full-fledged, efficient fault simulator which uses the hierarchical approach proposed by us over a basic sophisticated technique like the concurrent approach.

The design technique for easy diagnosability of regular circuits can be studied in more detail to understand the various types of masking of faults in case of multiple

numbers of them. A formal technique can be developed to come up with a higher confidence in such circuits.

Bibliography

[Avi75] A. Avizienis. Fault-tolerant systems. *IEEE Transactions on Computers*, December 1975.

[BF80] M. A. Breuer and A. D. Friedman. Functional level primitives in test generation. *IEEE Transactions on Computers*, pages 223–225, March 1980.

[BH90] D. Bhattacharya and J. P. Hayes. *Hierarchical Modeling for VLSI Circuit Testing.* Kluwer Academic Publishers, 1990.

[CJ95a] Dhruva R. Chakrabarti and A. Jain. Fault-diagnosis and tolerance in repetitive circuits. *Proceedings of the First International Conference on Fault-Tolerant Systems and Software*, pages 107–113, December 1995.

[CJ95b] Dhruva R. Chakrabarti and A. Jain. An improved hierarchical test generation technique for combinational circuits with repetitive sub-circuits. *Proceedings of the Fourth Asian Test Symposium*, pages 237–243, November 1995.

[CJ96] Dhruva R. Chakrabarti and A. Jain. An efficient test generation technique for sequential circuits with repetitive sub-circuits. *Proceedings of the Ninth International Conference on VLSI Design*, pages 174–177, January 1996.

[CR94] H. Cox and J. Rajski. On necessary and nonconflicting assignments in algorithmic test pattern generation. *IEEE Transactions on Computer-Aided-Design*, pages 515–530, April 1994.

[Dat85] *The TTL Data Book.* Texas Instruments Inc., 1985.

[ea88] Ma et al. Test generation for sequential circuits. *IEEE Transactions on Computer-Aided-Design*, October 1988.

[Eld59] R. D. Eldred. Test routines based on symbolic logic statements. *Journal of ACM*, March 1959.

[FS83] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, pages 1137–1144, December 1983.

[FSP85] M. Mezzalama F. Somenzi, S. Gai and P. Prinetto. Testing strategy and technique for macro-based circuits *IEEE Transactions on Computers*, pages 85–89, January 1985.

[Goe81] P. Goel. An implicit enumeration algorithm to generate tests for combinational circuits. *IEEE Transactions on Computers*, March 1981.

[KP93] W. Kunj and D. K. Pradhan. Accelerated dynamic learning for test generation in combinational circuits. *IEEE Transactions on Computer-Aided-Design*, pages 684–694, May 1993.

[KP94] W. Kunj and D. K Pradhan. Recursive learning : A new implication technique for efficient solutions to cad problems - - test, verification and optimization. *IEEE Transactions on Computer-Aided-Design*, pages 1143-1157, September 1994.

[MAF90] M. A. Breuer M. Abramovici and A. D. Friedman. *Digital Systems Testing and Testable Design.* Computer Science Press, New York, 1990.

[MH90] B T. Murray and J. P. Hayes. Hierarchical test generation using precomputed tests for modules. *IEEE Transactions on Computer-Aided-Design*, pages 594-603, June 1990.

[MMSS88] E. Trischler M. M. Schulz and Thomas M. Sarfert. Socrates : A highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided-Design*, pages 126–137, January 1988.

[Rot66] J. P. Roth. Diagnosis of automata failures : a calculus and a method. *IBM Journal of Research and Development*, October 1966.

[SA89] M. M. Schulz and E. Auth. Improved deterministic test pattern generation with applications to redundancy identification. *IEEE Transactions on Computer-Aided-Design*, pages 811–817, July 1989.

[SH81] T. Sridhar and J. P. Hayes. A functional approach to testing bit-sliced microprocessors. *IEEE Transactions on Computers*, pages 563–571, August 1981.

[TA80] S. M. Thatte and J. A. Abraham. Test generation for microprocessors. *IEEE Transactions on Computers*, pages 429–441, June 1980.